An Efficient Job-Grouping Based Scheduling Algorithm for Fine-Grained Jobs in Computational Grids

Arijit Mukherjee

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Orissa, India

An Efficient Job-Grouping Based Scheduling Algorithm for Fine-Grained Jobs in Computational Grids

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Computer Science)

by

Arijit Mukherjee

(Roll: 209CS1073)

under the supervision of

Dr. Pabitra Mohan Khilar

NIT Rourkela

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Orissa, India

May 2011

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Orissa, India.

Certificate

This is to certify that the work in the thesis entitled ***An Efficient Job-Grouping Based Scheduling Algorithm for Fine-Grained Jobs in Computational Grids*** by ***Arijit Mukherjee*** is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering with specialization in Computer Science. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: 3rd June 2011

Dr. Pabitra Mohan Khilar
Assistant Professor
Dept. of Computer Science and Engineering
National Institute of Technology, Rourkela
Orissa-769008

Acknowledgment

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis.

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Pabitra Mohan Khilar for providing me with a platform to work on challenging areas of grid computing. His valuable comments and suggestions were encouraging. He was the one who showed me the path from the beginning to the end.

I am also thankful to Prof. S. K. Rath, Prof. S. K. Jena, Prof. B. Majhi, Prof. A. K. Turuk, Prof. D. P. Mohapatra, Prof. R. Baliarsingh, and Prof. Bibhudatta Sahoo for giving encouragement and sharing their knowledge during my thesis work.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to help us in their respective roles.

I would like to thank all my friends and lab-mates for their encouragement and understanding. They made my life beautiful and helped me every-time when I was in some trouble.

Most importantly, none of this would have been possible without the love and patience of my family. My family to whom this thesis is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to them.

Arijit Mukherjee

Email: arijit.nitr@gmail.com

Abstract

Grid computing is a high performance computing environment to solve large-scale computational demands. Computational grids emerged as a next generation computing platform which is a collection of heterogeneous computing resources connected by a network across dynamic and geographically dispersed organizations, to form a distributed high performance computing infrastructure. In computational grid main emphasis is given on resource management and job scheduling. The main goal of scheduling is to maximize the resource utilization and minimize processing time of the jobs. Various research works has been done on job scheduling problem in grid, but still further analysis and research needs to be done to improve the performance of scheduling algorithm in computational grid. In this thesis, an efficient job-grouping based approach has been proposed for fine-grained job scheduling in computational grids. Resources in computational grid are heterogeneous in nature, owned and managed by different organizations with different allocation policies. In our scheduling algorithm jobs are scheduled based on resources computational and communication capabilities. Independent fine-grained jobs are grouped together based on the chosen resources characteristics, to maximize resource utilization and minimize processing time and cost. Job scheduling is a decision process by which application components are assigned to available resources to optimize various performance metrics. Hence in this thesis, we have specifically focused on improving computational grid performance in terms of makespan and total computation time. A simulation of proposed approach using GridSim toolkit is conducted. The performance of the algorithm is evaluated using above mentioned performance parameters. A comparison of our proposed approach with other existing fine-grained job scheduling strategies is provided. Experimental results show proposed algorithm performs efficiently in computational grid environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Grid Computing

Job Scheduling in Grid

Motivation

Thesis Organization

Chapter 1

Introduction

The computational speed of individual computers has rapidly increased in the past thirty years. However, they are still not faster enough for large-scale scientific problems. For example, The Large Hadron Collider (LHC) at CARN (European Organization for Nuclear Research) will produce 15 Petabytes (15 million Gigabytes) of data per year. These data requirements mean that most analysis programs need much more computational power than presently available supercomputers. This is why CERN and its partners in universities around the world are leading the development of the LHC Computing Grid (LCG) [1] which aims to link hundreds of major computing centers around the world. Therefore, in the mid 1990s Ian Foster and Carl Kesselman proposed a distributed computing infrastructure, which they called the grid [2]. The main objective behind the grid computing is to supply computing resources over the internet seamlessly, transparently and dynamically when needed, like the power grid supplies electricity to end users. A grid is a hardware and software infrastructure that provides a dependable, consistent, pervasive, and inexpensive access to high performance computing resources [3]. This technology is a type of distributed system which supports the sharing and coordinated use of resources, without considering location, in dynamic virtual organizations that shares the same goal [2]. The resource scheduling and management system is the central component of a grid system. Its basic responsibilities are to accepts jobs from users, and schedule the jobs to appropriate available grid resources [4]. However, grid performance can be improved in terms of job processing time by making sure all the resource available in the grid are utilized optimally using a good job scheduling algorithm. In traditional computing system,

job scheduling is well studied problem. Job scheduler are exists for many computing environments. These scheduler systems are designed to work under the assumption that they have complete control of a resource and thus can implement the mechanisms and policies needed for the effective use of that resource. Unfortunately, this assumption does not apply to the grid. When dealing with the grid we must develop methods for managing grid resources across separately administered domains, with the resource heterogeneity (grid resources are typically heterogenous) and differences in local policy [5].

1.1 Grid Computing

Grid is type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of self-service requirement [6]. A computational grid environment behaves like a virtual organization consisting of distributed resources. A virtual organization is a set of individuals defined by a definite set of sharing rules like what is shared, who is allowed to share, and the conditions under which the sharing takes place. The needs of the grid range from client-server to peer-to-peer architecture, from single user to multi-user systems, from sharing files to sharing resources, etc. and all these in a dynamic, controlled and secured manner. As grid computing focuses on dynamic and cross-organizational sharing, it enhances the existing distributed computing technologies. In most organizations, computing resources are under utilized. Most desktop machines are busy less than 25% of the time (if we consider that a normal employee works 7 hours a day and that 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these under utilized resources and thus has the possibility of substantially increasing the efficiency of resource usages. The easiest use of grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy, the execution of the task would be delayed. Grid computing should enable the job to be run on an idle machine elsewhere on the

network.

1.1.1 Types of Grid

Grid computing can be used in a variety of ways to address various kinds of application requirements. Grid systems are mainly classified on the basis of two factors: (1) Functionality and (2) Scale.

On basis of Scale grid can be classified as:

- **Cluster Grid:** Cluster grid is simplest form of grid and provides service to the group or department level.
- **Enterprise Grid:** Enterprise grids enable multiple group or department to share recourses with in enterprise.
- **Global Grid:** Global grids are collection of enterprise and cluster grid as well as other geographically distributed resources, all of which are agreed upon global usage policies and protocols to enable resources sharing.

On basis of Functionality grid can be classified as:

- **Computational Grid:** A computational grid is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified computing resource.
- **Data Grid:** Data grids primarily deal with providing services and infrastructure for distributed data-intensive applications that need to access, transfer and modify massive datasets stored in distributed storage resources[7].

1.1.2 Characteristics of Computational Grids

Computational grids are among the first type of grid systems. They were developed due to the need to solve problems that require processing a large quantity of operations or data. In spite of the fact that the capacity of the computers continues to improve, the computational resources do not respond to the continuous demand for

more computational power. Moreover, many statistical studies have shown that computers from companies, administration, etc. are usually underutilized. One of the main objectives of the computational grid is maximize the utilization of resources, to benefit from the existence of many computational resources through the sharing. Computational grid computing shares many of the same problems with both parallel an distributed computing, but there are some differentiating characteristics, and these characteristics are:

Heterogeneity:

There are three forms of heterogeneity. Obviously the hardware is heterogeneous. The system software (both libraries and operating systems) are also heterogeneous. Lastly, the administrative policies are heterogeneous. Since the resources will be owned by different organizations, there will be no single overarching policy that can be made to encompass all resources.

Dynamicity or Adaptability:

The resource pool (both in terms of hardware and software) that composes the grid will fluctuate over time. Even when the resource pool remains stable, grid resources are shared. As such, the performance that is available from various resources may fluctuate due to load. Finally, administrative and security policies may change as organizations grow or change their affiliations.

Scalability:

A grid might grow from few resources to millions. This raises the problem of potential performance degradation as a grids size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant.

Reliability and Management:

High performance conventional computing systems use expensive hardware to increase reliability. A grid is an alternate approach to reliability that relies more on software

technology than expensive hardware. The grid must deliver acceptable application performance to be effective. To do so, not only must it provide secure access to resources, but it must also support concurrent execution of cooperating application components.

1.1.3 Grid Architecture

Architecture identifies the fundamental system components, specifies purpose and function of these components, and indicates how these components interact with each other. Figure 1.1 shows the hardware and software stack within a typical grid architecture. It consists of four layers fabric, core middleware, user-level middleware and application and portals layers [6].

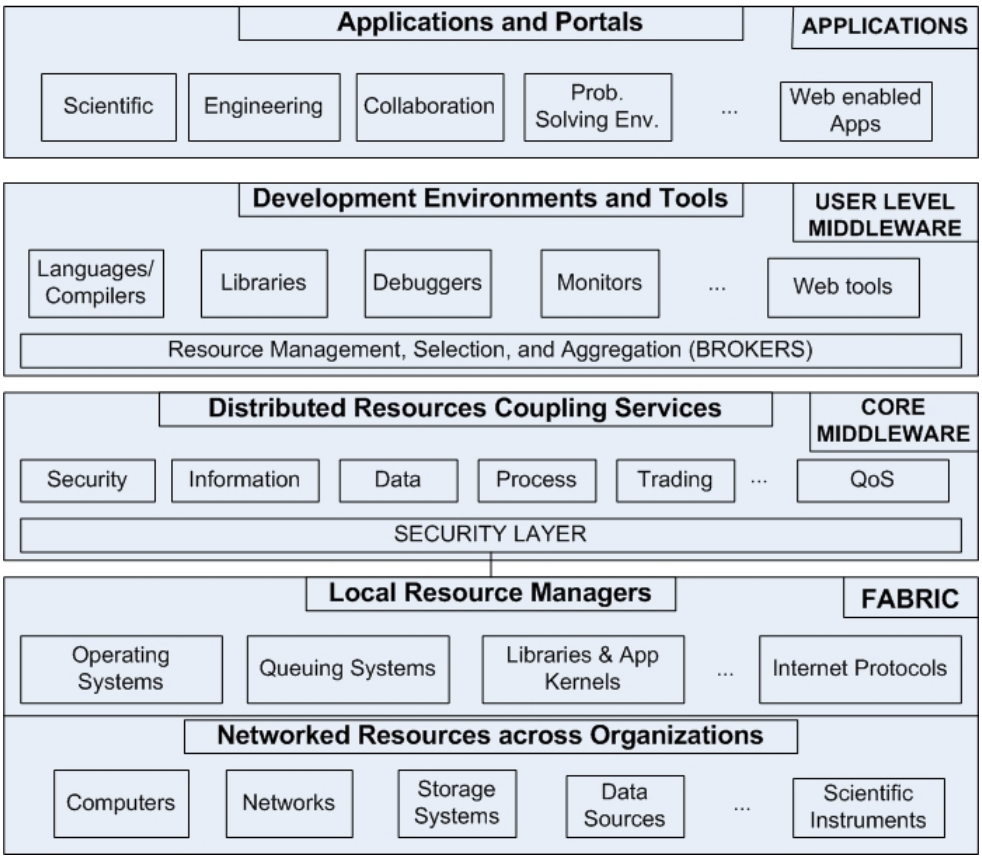


Figure 1.1: A Layered Grid Architecture and Components.

Grid Fabric: This layer consists distributed resources, networks and storage devices. The computational resources represent multiple architectures such as clusters,

supercomputers, servers and ordinary PCs which run a variety of operating systems.

Core Grid Middleware: This layer offers services such as remote process management, allocation of resources, storage access, resource information registration and discovery, security, and Quality of Service (QoS) such as resource reservation and trading. These services abstract the complexity and heterogeneity of the fabric level by providing a consistent method for accessing distributed resources.

User-level Grid Middleware: This layer utilize the interfaces provided by the core middleware to provide higher level abstraction and services. These includes application development environment, programming tools and resource scheduler for managing resources and scheduling application jobs for execution on global resources.

Grid Applications and Portals: The application and portals are developed using grid-enabled programming environments, interfaces and scheduling services provided by user-level middleware. Grid portals offer web-enabled application services, where users can submit and collect results for their jobs on remote resources through the web.

1.2 Job Scheduling in Grid

The scheduling problem in grid is one of the most studied problem in the research community. However, the grid setting there are several characteristics that make the problem different and more challenging than its version of conventional distributed systems. Some of these characteristics are dynamic structure of the computational grid, high heterogeneity of resources, jobs and interconnection networks, existence of local policies on resources and local schedulers, the large scale of the grid system and security [8].

In order to perform the scheduling process, the grid scheduler has to follow a series of steps [9]: (1) Collecting information of jobs submitted to the grid, (2) Collecting

available resource information, (3) Computation of the mapping of jobs to selected resources, (4) Jobs allocating according to the mapping, and (5) Monitoring of job completion.

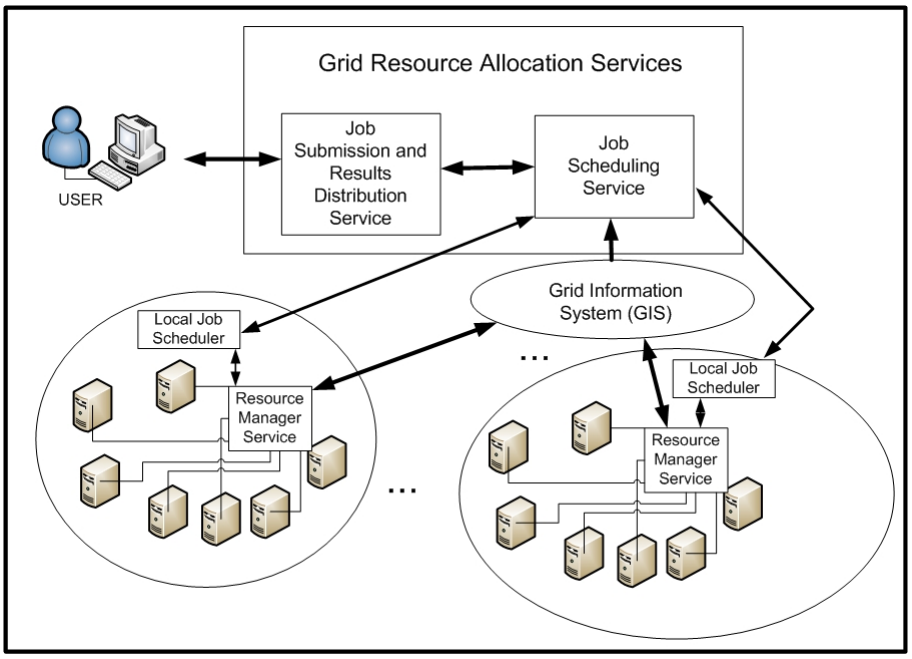


Figure 1.2: Basic Model of Grid System.

Different types of scheduling are found in grid system as applications could have different scheduling needs such as batch or immediate mode, task independent or dependent, uses local schedulers, centralized or decentralizes approach etc. In the following, we describe the main types of scheduling arising in computational grid environments.

Centralized, hierarchical and decentralized scheduling: Both centralized and decentralized scheduling are useful in grid computing. Essentially, they differ in the control of the resources and knowledge of overall grid system. In centralized scheduling, there is more control on resources, monitoring of the resource state, and therefore it is easier to obtain efficient schedule. This type of scheduling, however, suffers from low scalability and not appropriate for large-scale grids. Centralized schedulers have a single point of failure. In hierarchical scheduler, it allows one to coordinate different schedulers at a certain level. In this case,

schedulers at the lowest level in the hierarchy have knowledge of the resources. This scheduler type still suffers from fault tolerance, yet it scales better and is more fault tolerant than centralized schedulers. In decentralized or distributed scheduling there is no central entity controlling the resources. In decentralized schedulers, the local schedulers play an important role. The scheduling requests, either by local users or other grid schedulers, are sent to local schedulers, which manage and maintain the state of the job execution. This type of scheduling is more realistic for real grid systems of large scale, but it could be less efficient than centralized schedulers.

Static versus dynamic scheduling: There are essentially two main aspects that determine the dynamics of the grid scheduling, namely: (a) The dynamics of job execution, which refers to the situation when job execution could fail or, in the preemptive mode, job execution is stopped due to the arrival in the system of high priority jobs, and (b) The dynamics of resources, in which resources can join or leave the grid in an unpredictable way, their workload can significantly vary over the time, the local policies on usage of resources could change over the time etc. These two factors decide the behavior of the grid scheduler, ranging from static to highly dynamic.

Immediate versus batch mode scheduling: Immediate and batch scheduling are well-known methods, largely explored in distributed computing. In immediate mode, jobs are scheduled as soon as they enter the system, without waiting for the next time interval when the scheduler will get activated or the job arrival rate is small having thus available resources to execute jobs immediately. In batch mode, tasks are grouped in batches and scheduled as a group. In contrast to immediate scheduling, batch scheduling could take better advantage of job and resource characteristics in deciding which job to map to which resource since they dispose of the time interval between two successive activations of the scheduler.

Our proposed scheduling algorithm is dynamic, decentralized and we have used Immediate mode scheduling.

1.3 Motivation

Computational grids are used extensively for executing applications with computation intensive jobs, but there exist several applications with a large number of fine-grained jobs. This lightweight jobs involves high overhead time and cost in terms of job processing time at the grid resources and job transmission to and from grid resources. Therefore, there is a need for an efficient job grouping-based scheduling system to dynamically assemble the lightweight jobs of an application into a group of jobs, and send these grouped jobs to the grid resources. The dynamic grouping should be done based on the computational requirements of each job with respect to availability, processing capability and network bandwidth of computational grid resources. N. Muthuvelu, *et. al.* [10] was the first to provide a job-grouping based scheduling strategy for applications with lightweight jobs. However, the algorithm there are some problems, the algorithm dose not considered dynamic resource characteristics, resource network bandwidth and it can not utilize resources sufficiently. In [11] Ng Wai Keat, *et. al.* and [12] Liu-Liao, they taken network bandwidth into account for scheduling, but they dose not pay attention to the dynamic characteristics of resources and sufficient utilization of the resources.

In this thesis, we proposed an efficient Dynamic Bandwidth-Aware Job-Grouping Based Scheduling algorithm to solve the problems mentioned above.

1.4 Thesis Organization

The organization of rest of the thesis and a brief outline of the chapters in this thesis is as follows.

In chapter 2, some related works on job scheduling in computational grids and their merits and demerits have been discussed.

In chapter 3, we have described our proposed approach for fine-grained job scheduling in grid, with details of scheduling model, architecture of the schedular and the scheduling algorithm.

Chapter 4, focusses on the setup of the simulation and experimental results. Performance criteria and environmental settings are introduced in this chapter, and a

thorough comparison of the performance of the algorithm with other approaches is provided.

Finaly, chapter 5 is dedicated to conclusion and future work.

Chapter 2

Related Work

Related Work on Job Scheduling in Grid

Summary

Chapter 2

Related Work

This chapter will provide a brief literature survey of different job scheduling schemes in computational grid.

2.1 Related Work on Job Scheduling in Grid

There are several algorithms and approaches in the area of job scheduling in in grid. A grate number of scheduling strategies have been developed to bring grid resource management and job scheduling issues to a more advanced level of efficiency. Some of the current job scheduling algorithms are: Minimum Execution Time, in this method minimum execution time is used to assign the job without considering the resource availability. Job is assigned to the resource on which it can be executed in minimum time. Allocation job without considering resource availability results less resource utilization [13]. Minimum Completion Time, in this approach job is assigned to the resource that gives minimum completion time (ready time of resource + job execution time on the selected resource) for the job. In this method also the less faster resources will stay under utilized [13]. Min-Min, in this method completion time of all unassigned task ($1 \leq i \leq n$) on all the available resources ($1 \leq j \leq r$) is used to calculate the minimum completion time ($MCT_{i,j}$) of Task T_i on the resource R_j. Then task which gives minimum of ($MCT_{i,j}$) is identified and assigned on the resource R_j. Subsequently the task T_i is removed from the list of unassigned task and workload of the resource M_j is updated. Above procedure is repeated until unassigned task list is empty [14]. Max-Min, This method is similar to min-min

method, completion time of all the unassigned tasks ($1 \leq i \leq n$) on all the available resources ($1 \leq j \leq r$) is used to calculate the minimum completion time ($MCT_{i,j}$) of Task T_i on the resource R_j. Then task which gives maximum of ($MCT_{i,j}$) is identified and assigned on the resource R_j. Subsequently the task T_i is removed from the list of unassigned task and workload of the resource M_j is updated. Above procedure is repeated until unassigned task list is empty [15]. Population-based heuristic methods, Mainly the following population-based heuristic methods are used in Job scheduling in grid: Genetic Algorithm [16], Ant Colony Optimization [17] and Particle Swarm Optimization [18].

These above scheduling optimization methods dose not considered characteristics of the fine-grained jobs, [10] shows that this lightweight jobs can cause high overhead in task processing time. Taking into account the characteristics of lightweight jobs, there are some research on lightweight job scheduling problem.

In [10], N. Muthuvelu, *et. al.* presents a dynamic job grouping-based scheduling algorithm that groups the jobs according to processing power of the available resource. It selects resources in first come first serve order and multiplies the granularity size to increase the resource computation time. Then it selects jobs and assign to groups job in first come first serve order and compare to resource if the grouped job's processing requirement is less than resource processing power (with in the granularity size), then assign another job and this process continues until the resource processing power is less to group job and remove very last job from the group and stop the grouping procedure. When all the jobs are grouped , it sends all the job groups to their corresponding resource. The algorithm reduces the total processing time and cost of the jobs, it also maximizes the resource utilization. However , the algorithm dose not take the dynamic resource characteristics into account and it dose not pay attention to the network bandwidth of the resources.

In [11], N.W. Keat, *et. al.* proposed a bandwidth-aware job-grouping based scheduling algorithm. It schedules jobs in grid systems by taking into consideration of computational capabilities and the communication capabilities of the resources. It uses network bandwidth of the resources to determine the priority of each resource.

The scheduler select the available highest priority resource and groups independent lightweight jobs together based on chosen resources processing capability. These jobs are grouped in such a way to maximize the utilization of resources. After grouping all jobs sends to the corresponding resources. This algorithm minimize the network latency, but the scheduling strategy is not ensuring that the resource having a sufficient bandwidth to send the group jobs within required time.

In [19], T.F. Ang, *et. al.* presents a grouping based bandwidth-aware scheduling strategy that schedules the jobs by taking into consideration of their computational capabilities and the communication capabilities of the resources, same as [11]. But the job groups are sent to the corresponding resources based on Largest Job First (LJF) strategy and the results of the processing are sent back to the user after they have been computed at their respective resources. The principle behind the bandwidth-aware scheduling is the scheduling priorities taking into consideration the communication capabilities of the resources. This dose not considered the dynamic resource characteristics into account and the scheduling strategy is not ensuring that the resource having a sufficient bandwidth to send the group jobs within required time same as above approach.

A grouping-based fine-grained job scheduling model is presented in [12] by Liu and Liao, the fine-grained jobs grouped into forming coarse-grained are allocated to the available resources according to their processing capabilities and network bandwidth in Largest Job First(LJF) order. but here resource are selected in FCFS order, there is no priority for selecting resources.

2.2 Summary

The objective of this thesis is to present an efficient job scheduling approaches for lightweight jobs in computational grid, which is based on previous research efforts in grouping-based job scheduling. Therefore, this chapter we gave overview of variety of existing techniques related to fine-grained job scheduling for grid.

Chapter 3

Fine-grained Job Scheduling in Computational Grids

Problem Definition

Job Scheduling Model

Architecture of Job Schedular

Algorithm Description

Summary

Chapter 3

Fine-grained Job Scheduling in Computational Grids

3.1 Problem Definition

We assumed decentralized computational grid system model with heterogeneous resources and user applications with large number of lightweight jobs. In this thesis, we discuss the problem of scheduling n jobs on m computing resources with an objective of minimizing the processing time and utilizing the resource effectively. If the number of job is less then the resources in the grid then the jobs can be allocated on the suitable resources. If the number of jobs is more than the number of resources, then an efficient scheduling algorithm is required for allocation of jobs. So, we considering number of jobs is more than the computing resources in this thesis. One job can not be assigned to different resources, no job migration is allowed.

To formulate the problem , lets define T_i where $i = 1, 2, ..., n$ as n independent jobs and R_j where $j = 1, 2, ..., m$ as m resources. Processing time for job i on computing resource j is $P_{i,j}$. $TP(S)$ represents the total processing time of n jobs on m computing resources. The main objective is find a efficient scheduling strategy S, which minimizes the total processing time:

$TP(S) = \sum_{i=1}^{n} \sum_{j=1}^{m} P_{i,j} * X_{i,j}$ Where, if job T_i scheduled to R_j in S then $X_{i,j} = 1$ else $X_{i,j} = 0$.

We proposed a dynamic job grouping based approach to find minimal processing time and effective resource utilization.

3.2 Job Scheduling Model

The four basic building blocks of grid model are user, job schedular, Grid Information System (GIS) and resources. User jobs submitted to the grid schedular for scheduling to the resources with an objective of minimizing the processing time and utilizing the resources effectively. The scheduling framework illustrated in Fig. 3.1 depicts the design of the job scheduler and its interactions with other entities. The job scheduler is a service that resides in a user machine. When the user creates a list of jobs in the user machine, these jobs are sent to the job scheduler for scheduling. The job scheduler obtains information of available resources from the Grid Information Service (GIS). Based on this information, the job scheduling algorithm is used to grouping the jobs and then resource selection for grouped jobs. When all the jobs are put into groups with selected resources, the grouped jobs are dispatched to their corresponding resources for computation by the dispatcher.

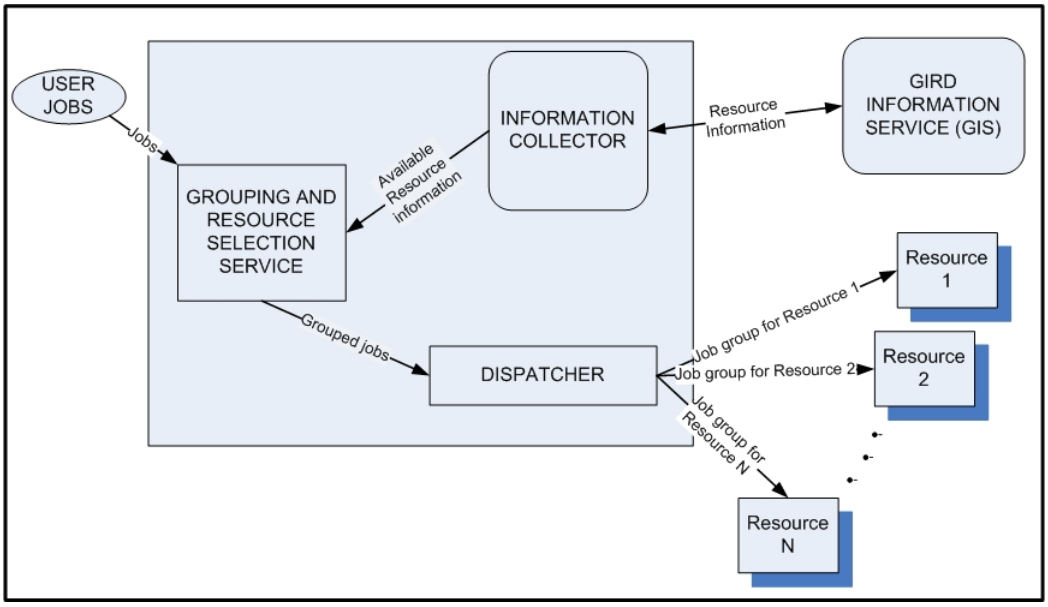


Figure 3.1: Scheduling Model for Job Grouping.

The Grid Information Service (GIS) provides information about all the registered resources in a grid. This service keeps track of all of the resources characteristics in the grid. GIS collects resource characteristic information like operating system, system architecture, processing capability, network bandwidth and processing cost.

It also provides users the availability information of the resources.

The information collector collects information from the Grid Information Service (GIS). It assembles the resource availability and processing capability to the resource information table. It also gathers information of the network bandwidth and processing cost of each listed resource provided by the GIS. The information collector is used by the grouping and resource selection service to gather necessary information to perform job grouping.

The grouping and resource selection service is responsible for grouping of job based on information collected by the information collector from GIS. In the job grouping process, user submitted jobs are collected by scheduler and jobs are grouped based on the selected available resource characteristics. The process iteratively performed until all the jobs are grouped according to corresponding resources.

The dispatcher acts as a sender that sends grouped jobs to their respective resources. The dispatcher forwards the grouped jobs based on the schedule made by the grouping and resource selection service. The dispatcher also collects the results of the completed jobs from the resources.

3.3 Architecture of Job Scheduler

The architecture of the job scheduler system is described in Fig. 3.2. The system accepts jobs from the grid users specified by their JOB_ID, JOB_LENGTH (in Million Instructions(MI)), JOB_INPUT_FILE_SIZE (in Mb) and total number of jobs submitted by the user. After gathering details of user jobs, schedular collects all the available computational grid resources information specified by their RESOURCE_ID, RESOURCE_MIPS (computational power of the resource in Million Instructions Per Second), RESOURCE_BANDWIDTH (in Mb/sec.), RESOURCE_COST (in cost/sec.).

After gathering the details of user jobs and the available resources, the scheduler will select a resource and multiplies the resource MIPS with the given granularity time [10], which is the time within which a job is processed at the resource. The value of this calculation produces the total Million Instructions (MI) for that particular resource to process within a particular granularity time. The system selects jobs in

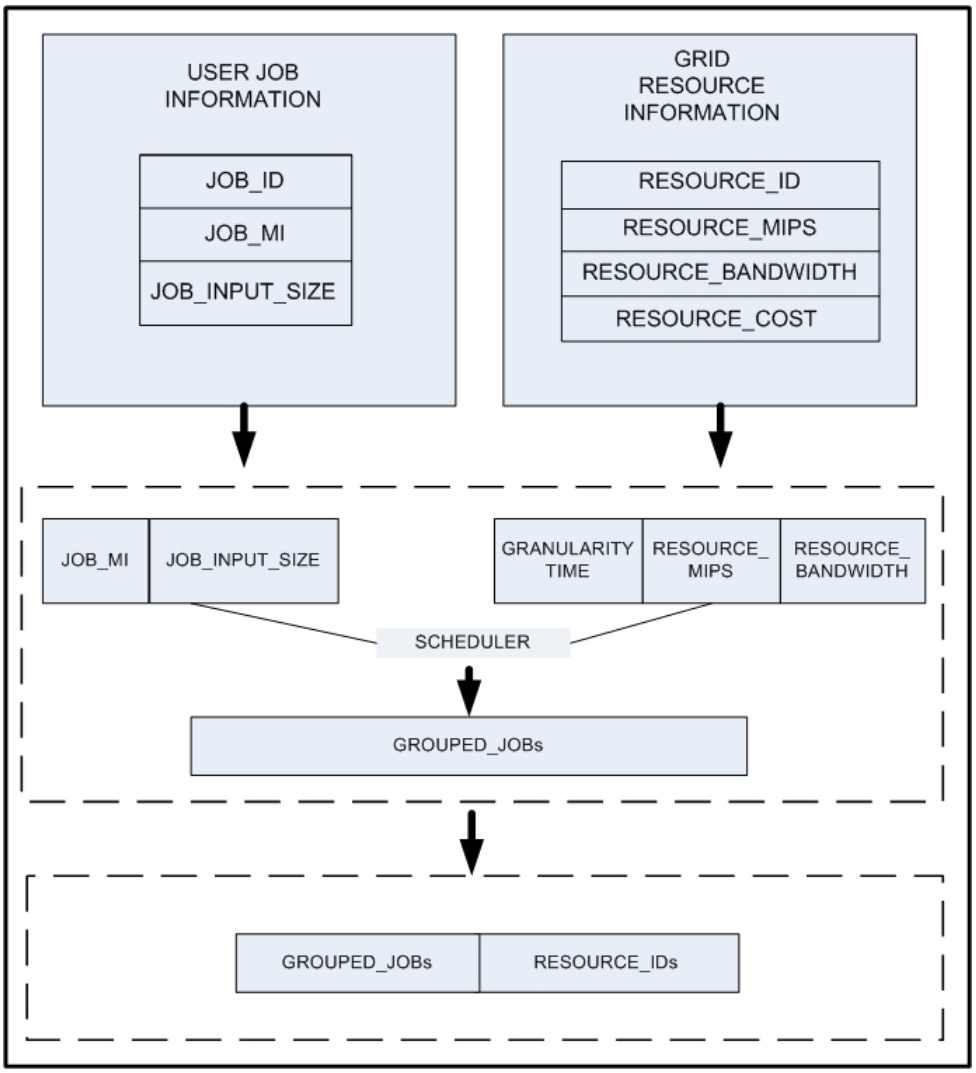


Figure 3.2: Architecture of Job Schedular.

first-come first-serve (FCFS) order, then jobs are grouped based on the resulting total MI of resource and bandwidth. New IDs are assigned to grouped jobs and scheduler submits the job groups to their respective resources for computation. After executing the group job, results goes to back to the corresponding users and the resource is again available to scheduler system.

3.4 Algorithm Description

In this section we present a job grouping-based fine-grained job scheduling algorithm. The algorithm mainly consists of two phases: (1) Create available resource list and (2) Job grouping and scheduling. The jobs are described as: JOB_i =

$\{JOB\_ID_i, JOB\_MI_i, JOB\_INPUT\_SIZE_i\}$ Resources are described as: $RES_j = \{RES\_ID_j, RES\_MIPS_j, RES\_BW\_RATE_j, RES\_COST_j\}$ After grouping job new group jobs will be created, described as:

$GroupedJOB_k = \{GroupedJOB\_ID_k, GroupedJOB\_MI_k, GroupedJOB\_INPUT\_SIZE_k\}$.

Notation	Definition
JOB_ID_i	Job ID of job i
JOB_MI_i	Job length of job i (in MI)
$JOB\_INPUT\_SIZE_i$	Input file size of job i (in Mb)
RES_ID_j	Resource ID of resource j
RES_MIPS_j	Processing capability of resource j (in MIPS)
$RES\_BW\_RATE_j$	Network bandwidth of resource j (in Mb/sec.)
RES_COST_j	Processing cost of resource j (in cost/sec.)
$GroupedJOB\_ID_k$	Assigned ID of grouped job k
$GroupedJOB\_MI_k$	Total job length of grouped job k
$GroupedJOB\_INPUT\_SIZE_k$	Total input fie size of groped job k
$Granularity\_Time$	Granularity time for the job grouping activity
RES_LIST	List of available grid resources
JOB_LIST	User Job list waiting to schedule
JOB_LIST_1	List not scheduled jobs after one iteration
$GROUPEDJOB\_LIST$	List of job groups after job grouping activity

Table 3.1: Notation Symbols and Their Definitions.

Grouping of job is done based on the processing capabilities and network bandwidth of available resources. First user submitted jobs are collected and one job list will be created, then scheduler collects available resource characteristics. Schedular select resource for scheduling job based on following conditions:

$$GroupedJOB\_MI_j + JOB\_MI_i < RES\_MIPS_j * Granularity\_Time \text{.....(1)}$$

$$(GroupedJOB\_MI_j + JOB\_MI_i)/RES\_MIPS_j < Granularity\_Time \text{.....(2)}$$

Equation (1) specifies that the processing requirement of the grouped job should not exceed the resource processing capability within a specified granularity size, equation (2) the total transfer time of the grouped jobs should not exceed total processing time of the group jobs. This two are the main constraints in job grouping strategy to achieve minimum job processing time and high resource utilization in computational grid system.

After collecting user jobs scheduler will create a JOB_LIST and available resource information will be gathered using following algorithm:

Algorithm 1 Listing of the Available Resources

1: Create a List RES_LIST.
2: Collect all available resource characteristics and add them to RES_LIST.
3: Sort the RES_LIST in descending order with respect to available processing capability (in MIPS).

After listing resources following algorithm will be used for job grouping and scheduling:

Algorithm 2 Job Grouping and Scheduling

1: **while** $JOB\_LIST.SIZE$!= 0 **do**
2: **for** $i = 0 \rightarrow JOB\_LIST.SIZE - 1$ **do**
3: **for** $j = 0 \rightarrow RES\_LIST.SIZE - 1$ **do**
4: **if** $(((GroupedJOB\_MI_j + JOB\_MI_i) < (RES\_MIPS_j * Graularity\_Time))$ and $(((GroupedJOB\_INPUT\_SIZE_j + JOB\_INPUT\_SIZE_i)/RES\_BW\_RATE_j) < Graularity\_Time))$ **then**
5: assign the JOB i to GroupedJOB j;
6: update JOB_i status to assigned with RES_ID_j;
7: break;
8: **end if**
9: **end for**
10: **if** JOB_i is not assigned **then**
11: Add the JOB_i to JOB_LIST_1;
12: **end if**
13: **end for**
14: **for** $k = 0 \rightarrow RES\_LIST.SIZE - 1$ **do**
15: **if** GroupedJOB k is not empty **then**
16: Create a new job with all the GroupedJob assigned to RES_k;
17: Assign a unique $GroupedJOB\_ID$ to newly created job and add it to $GroupedJOB\_LIST$;
18: Allocate the $GroupedJOB_k$ to RES_k;
19: **end if**
20: **end for**
21: **if** $JOB\_LIST_1.SIZE$!= 0 **then**
22: $JOB\_LIST = JOB\_LIST_1$;
23: Update resource information using algorithm 1;
24: **end if**
25: **end while**
26: **for** $i = 0 \rightarrow GroupedJOB\_LIST.SIZE - 1$ **do**
27: Receive computed $GroupedJOB_i$ from respective resources;
28: **end for**

The overall explanation of Algorithm 2 is as follows: once the scheduler gathers the characteristics of the available grid resources using algorithm 1. Then, schedular selects a job i from JOB_LIST and available resource j from RES_LIST, if job i satisfies the conditions (1) and (2), then scheduler add job i to Job Group j, else add job i to JOB_LIST_1. This process continues until JOB_LIST is empty. Now all the user jobs (which are in JOB_LIST_1) are grouped and assigned to the grid resources. The scheduler then sends the job groups to their respective resources for computation. After that scheduler will check JOB_LIST_1, if list is not empty then all the jobs form JOB_LIST_1 will be added to JOB_LIST. The scheduler will get resource status using Algorithm 1. This steps will be repeated to schedule remaining jobs. The grid resources process the received job groups and send back the computed job groups to the scheduler. The scheduler then gathers the computed job groups and split the output before sending to the user.

3.5 Summary

In this chapter, in order to minimize job processing time and utilize grid resources sufficiently, an efficient dynamic job grouping based job scheduling algorithm has been proposed. This grouping based fine-grained job scheduling strategy has taken into account the computational and communication capabilities of available grid resources to schedule the jobs. The complexity of the sorting of resources according to their available MIPS is $O(nlogn)$, without which the scheduling algorithm will run in linear time.

Chapter 4

Implementation and Results

Implementation Details

Results

Summary

Chapter 4

Implementation and Results

We simulated the proposed algorithm using GridSim toolkit [20] to verify the improvement of the proposed approach over other scheduling strategies. In this chapter, we have discussed implementation details of the simulation environment. The experimental results are analyzed based on performance criterions.

4.1 Implementation Details

4.1.1 GridSim Toolkit

The GridSim toolkit used to simulate grid environment is a Java based grid simulation tool. This toolkit supports modeling and simulation of heterogeneous grid resources. It also provides Application Programming Interface(API) [21] for creation of user jobs, resource management and job schedulers[20].

The GridSim toolkit supports simulation of a wide range of heterogeneous resources, such as clusters with different configuration. It can be single or multiprocessor with different processing capability, network bandwidth, operating system, system architecture etc. The GridSim toolkit was chosen to simulate our scheduling algorithm, because of the following features:

- It allows modeling of heterogeneous types of resources.
- Resource capability can be defined (in the form of MIPS as per SPEC benchmark).
- Application Jobs can be heterogeneous and they can be CPU or I/O intensive.

- There is no limit on number of jobs can be created and submitted to a resource.
- Network speed of the resources can be specified.
- Statistics of all operations can be recorded. These statistics can be used for further analysis.

4.1.2 System Model

For experimental purposes we assume that the grid consists of 10 resources. In general, each resource contains 3 computing nodes (Machines), and each computing node contains 4 Processing Elements (PE). The processors of computing nodes in different resources have different processing power (in MIPS). Network speed among the computing nodes are also different for different resources. The characteristics of 10 resources used in our experiment, shown in Table 4.1. In GridSim jobs can be executed at the resources in a space-shared policy or a time-shared policy. The time-shard policy (Round-robin) enables all arriving tasks to start immediately as they arrive at a resource, in the space-shared policy jobs are executed in First-come, first-serve (FCFS) order. In our experiment, we used space-shared policy.

Resource	**MIPS of each PE**	**Total MIPS**	**Cost/unit time**	**Network (Bandwidth) (in Mb/unit time)**
R1	50	600	150	330
R2	39	468	130	300
R3	72	864	120	210
R4	60	720	200	400
R5	20	240	110	300
R6	120	1440	250	300
R7	90	1080	175	330
R8	24	288	200	320
R9	40	480	100	210
R10	66	792	145	150

Table 4.1: Grid resource characteristics.

4.1.3 Application Model

In our application model, we assume that jobs which are submitted to the grid are independent tasks with no required order of execution. The jobs are of different computational size, each job also have different input files size requirements. The computational requirement (Job length) of each job is presented in Millions Instructions (MI). In GridSim, jobs are created and their requirements are defined through Gridlet objects [20]. A Gridlet is a package that contains all the information related to the job and its execution management details such as the job length (in Million Instructions(MI)), the size of input files, and the job originator (user information). The characteristics of the Gridlet sent to the grid to compare the performance of different algorithms are shown in 4.2.

Gridlet characteristics	
Length	1 - 100 MI (with 50 MI average)
Input file size	30 + (with 10% to 40% random variation) MB
Output file size	35 + (with 10% to 50% random variation) MB

Table 4.2: Gridlet characteristics.

4.1.4 Performance Evaluation Criteria

In this section, we define our performance evaluation criteria which are used to evaluate the performance of our algorithm. The criteria includes makespan, computation time, processing cost and resource utilization. We have used an average of ten runs in order to account realistic conditions.

Makespan

One of the most common measures in evaluating the performance of a scheduling algorithm is the makespan. The makespan is the total application execution time. The total application execution time is measured from the time the first job is sent to the grid, until the last job comes out of the grid. The makespan includes the total

computation time taken at the resource, total time to send the jobs to resource and receiving results from resources, and time taken to schedule the jobs. In real world, the makespan of an application depends on the current network load and speed. Makespan is an indicator of the general productivity of the grid system, small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources.

Computation Time

Total computation time (C_t) is the sum of total time taken by each resource (R_i) to compute the assigned job.

$C_t = \sum_{i=1}^{m} C_{R_i}$,

where, m = total number of resource and C_{R_i} = total computation time of assigned jobs at resource R_i.

Processing Cost

The total processing cost is computed based on the actual computation time taken for computing the jobs at the grid resource and at the cost rate specified at the Grid resource, as summarized below:

$ProcessingCost = C_{R_i} * P_{R_i}$, where

C_{R_i} = total computation time of assigned jobs at resource R_i and P_{R_i} = Cost per unit time of the resources R_i.

Resource Utilization

Maximizing the resource utilization of the Grid system is another important objective. Resource utilization in terms of computational capability can defined as follows:

$$\text{Resource Utilization} = \frac{TotalProcessingLoadAtTheResource}{TotalProcessingPowerOfTheResource} * 100$$

4.2 Results

In order to evaluate the performance of our algorithm, a set of experiments are conducted to measure makespan, total computation time, processing cost and resource utilization. We compared the results of our proposed Dynamic Bandwidth-Aware Job-Grouping Based Scheduling [DBAJGBS] algorithm with DJGBS [10], BAJGBS [19] and GFJS [12] algorithms. In the simulation, we performed scheduling experiments by setting different values to the number of jobs, the number of job is varied from 100 to 1000 and granularity time for the job grouping activity is defined as 15 time unit. The tables 4.3, 4.4 and 4.5 shows the simulation results of DBAJGBS, DJGBS, BAJGBS and GFJS algorithms. The Figures 4.1, 4.2 and 4.3 depicts, the makespan, total computation time and cost are minimized by using DBAJGBS algorithm compared to DJGBS, BAJGBS and GFJS algorithms.

Number of Jobs	**DBAJGBS**		**DJGBS**		**BAJGBS**		**GFJS**	
	M	NG	M	NG	M	NG	M	NG
100	291	1	376	2	283	1	370	1
200	438	2	509	3	514	1	517	3
300	549	3	660	4	618	2	632	3
400	730	4	840	4	719	3	731	4
500	858	5	951	4	836	3	839	5
600	939	5	1040	6	914	3	969	6
700	1040	6	1128	6	1152	3	1054	7
800	1165	7	1292	6	1258	5	1167	8
900	1301	8	1546	6	1370	6	1345	9
1000	1411	9	1801	7	1482	7	1472	9

Table 4.3: Comparison of makespan for different numbers of jobs with average MI of 50 and granularity time of 15 time unit. (NG = Numbers of job groups created, M = Makespan (in time unit))

4.2.1 Makespan

Figure 4.1 shows a comparison between the makespan of the proposed algorithm and other existing algorithms. The experimental results show DBAJGBS is performing

best amongst all. From the figure 4.1, it is observed that the makespan of DBAJGBS and GFJS are almost same for 400, 500, 600, 700 and 800 numbers of job, but over all performance of DBAJGBS is better.

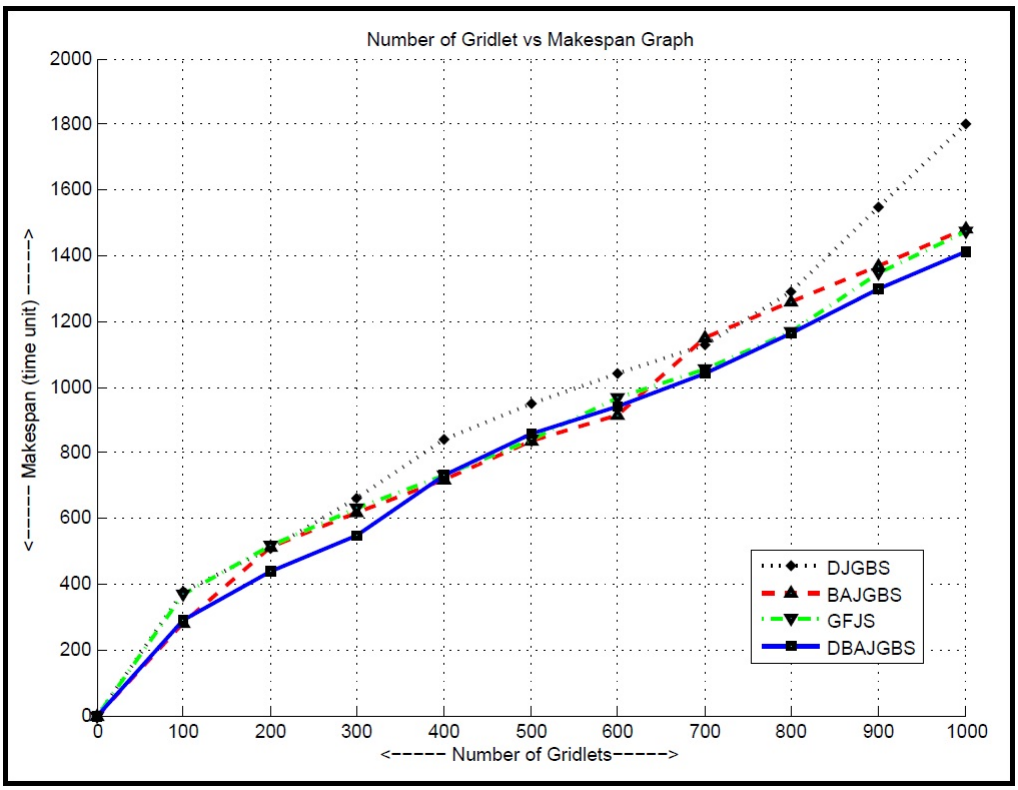


Figure 4.1: Comparing the makespan of different approaches with granularity time of 15 time unit.

4.2.2 Computation Time

The figure 4.2 depicts the total computation time taken by the resources to compute the assigned jobs for all the algorithms. The BAJGBS algorithm gives best performance along the existing algorithms, but after 700 jobs the computation time is increased rapidly. In case our proposed algorithm (DBAJGBS), it reduces the total cmputation time by 6% - 42% when compared to BAJGBS algorithm depending on the number of jobs. The graph shows that the proposed scheduling algorithm is able to operate in conditions of different number of jobs with good improvement over an existing job grouping based scheduling algorithm.

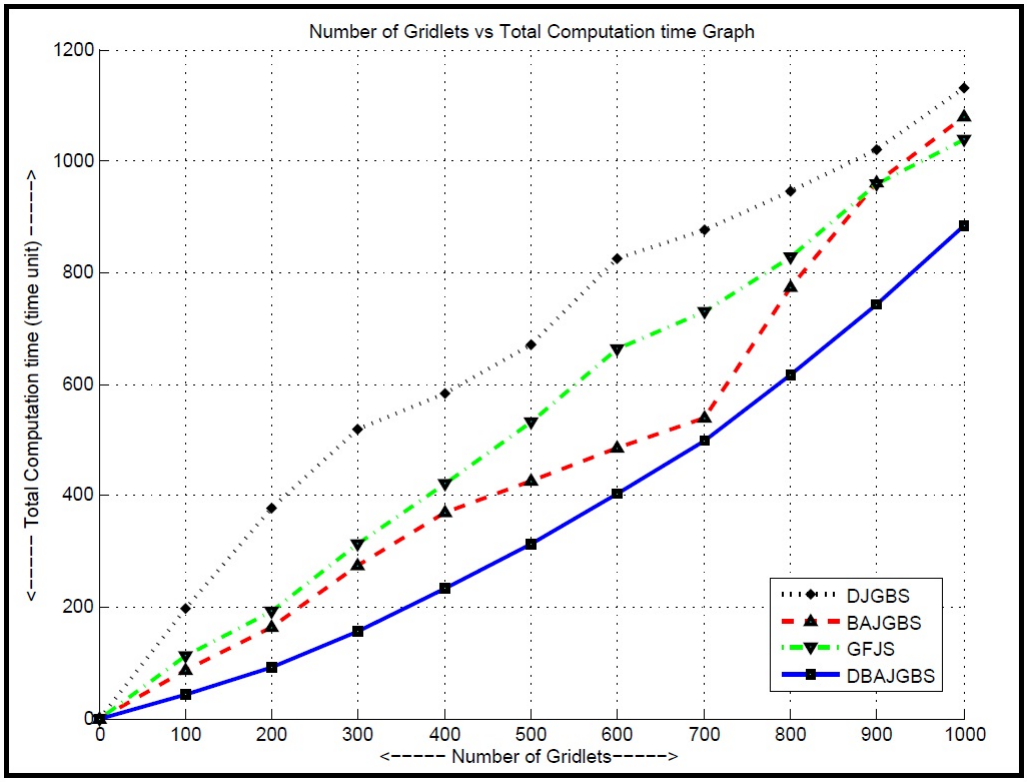


Figure 4.2: Comparing the total computation time of different approaches with granularity time of 15 time unit.

Number of Gridlets	**DBAJGBS** Computation Time	**DJGBS** Computation Time	**BAJGBS** Computation Time	**GFJS** Computation Time
100	43	197	85	113
200	91	376	164	192
300	157	518	273	314
400	232	583	370	421
500	314	672	426	533
600	403	826	485	664
700	499	876	540	729
800	617	947	773	828
900	744	1022	961	959
1000	884	1131	1080	1039

Table 4.4: Comparison of total computation time for different numbers of jobs with average MI of 50 and granularity time of 15 time unit.

4.2.3 Processing Cost

In terms of processing cost, the total time each Gridlet spends at the grid resource of computation is taken into account for calculating the processing cost. The figure 4.3 depicts the graph of the results collected from the simulations. The DBAJGBS algorithm select the resource based on their available computational capability it reduces the total computation time for that reason it also reduce the processing cost. However, we did not considered optimization of the processing cost in our experiments. Some cost-optimization can be used to decrease the total cost.

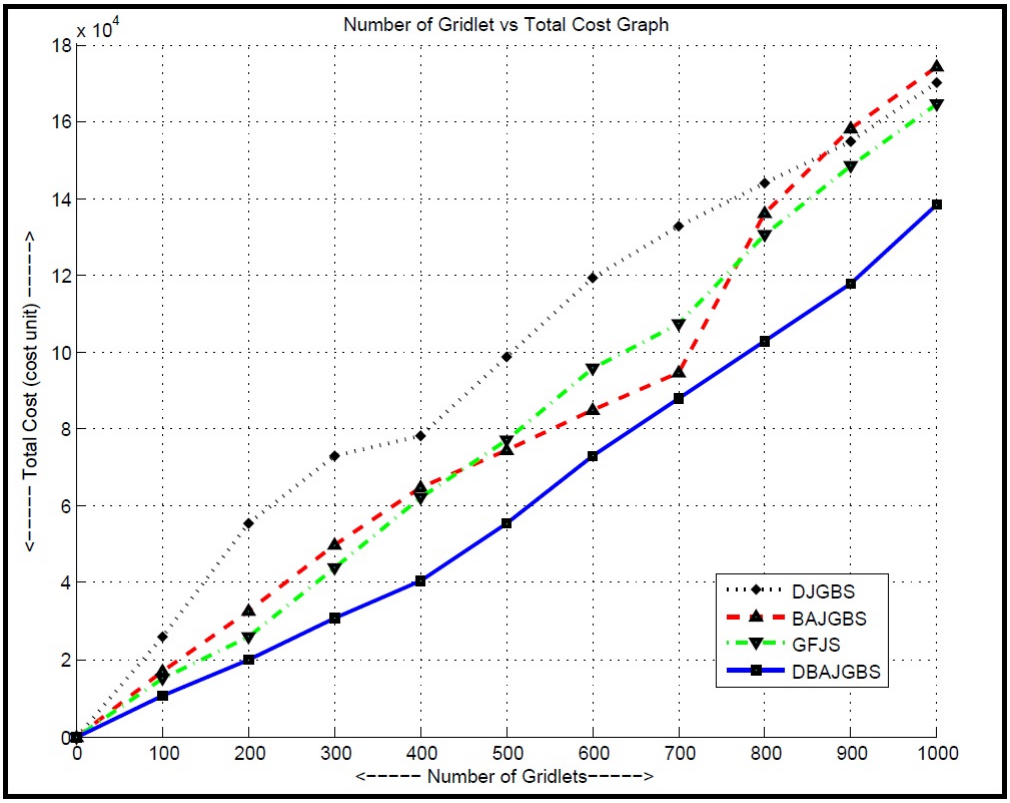


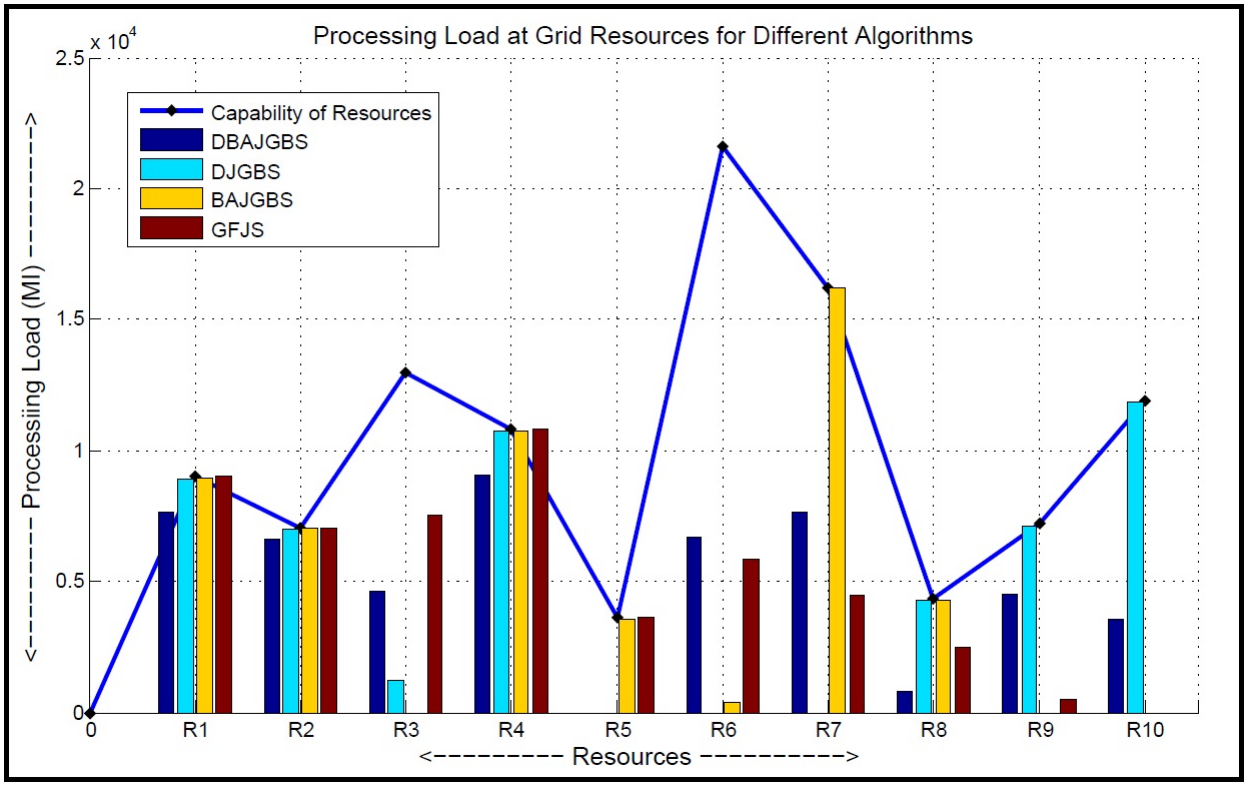
Figure 4.3: Comparing the processing cost of different approaches with granularity time of 15 time unit.

Number of Gridlets	DBAJGBS Cost	DJGBS Cost	BAJGBS Cost	GFJS Cost
100	10750	26134	17000	14984
200	20125	55579	32800	26004
300	30795	72836	49950	44004
400	40445	78171	64750	62070
500	55305	98870	74550	77126
600	73105	119342	85050	95797
700	87955	132946	94500	107504
800	103055	144008	136060	130425
900	117705	154895	158040	148381
1000	138285	170282	174110	164496

Table 4.5: Comparison of total processing cost for different numbers of jobs with average MI of 50 and granularity time of 15 time unit .

4.2.4 Resource Utilization

Figure 4.4 shows the load at the resources when different algorithms are used to schedule 1000 Gridlets with an average length of 50 MI. Granularity time is taken as 15 time unit. When DJGBS algorithm is used seven resources are utilized: R1(99%), R2(99%), R3(9.6%), R4(98%), R8(98.9%), R9(99%) and R10(99%). The resources are selected in FCFS order by DJGBS algorithm. In case of BAJGBS algorithm, it selects resources based on the network bandwidth, utilized seven resources: R1(99%), R2(99%), R4(99%), R5(98.8%), R6(2%), R7(99%) and R8(99.3%). Nine resources are utilized by the GFJS algorithm, the resources are selected in first-come first-serve [FCFS] order: R1(99.9%), R2(99.9%), R3(58%), R4(99.9%), R5(99.9%), R6(26.9%), R7(27.5%), R8(57.6%) and R9(7%). The DBAJGBS algorithm utilized nine resources: R1(84.9%), R2(94%), R3(35.7%), R4(84.3%), R6(51.5%), R7(78.6%), R8(18.79%), R9(62.9) and R10(29.9%).



Standard deviation of utilization of resources by using GFJS algorithm is 40.6. In our proposed DBAJGBS algorithm the standard deviation is 32. From the analysis of simulation results we can say DBAJGBS utilize resources effectively then other existing fine-grained job scheduling algorithms.

4.2.5 Simulation with different granularity time

Simulations are conducted using different granularity time to analyze makespan and total computation time taken to complete execution of 1000 Gridlets. Table 4.6 and figure 4.5 depict the results of simulations carried out using different granularity time. From the experimental results, it is observed that job grouping method decreases the total computation time, how ever assigning a large number of jobs to one particular resource will increase the makespan.

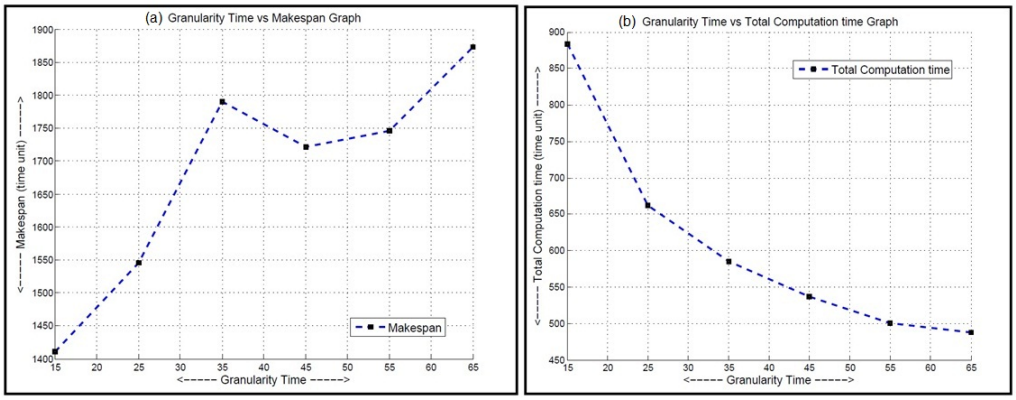


Figure 4.5: (a) Makespan and (b) Total computation time for executing 1000 jobs using different granularity time.

Granularity Time	**15**	**25**	**35**	**45**	**55**	**65**
Computation Time	884	662	585	537	501	488
Processing Cost	138285	119755	100939	100530	103275	103850
Makespan	1411	1546	1790	1721	1746	1873
Number of Groups	9	5	4	3	2	2

Table 4.6: Simulation results of DBAJGBS algorithms with different Granularity Time for 1000 jobs.

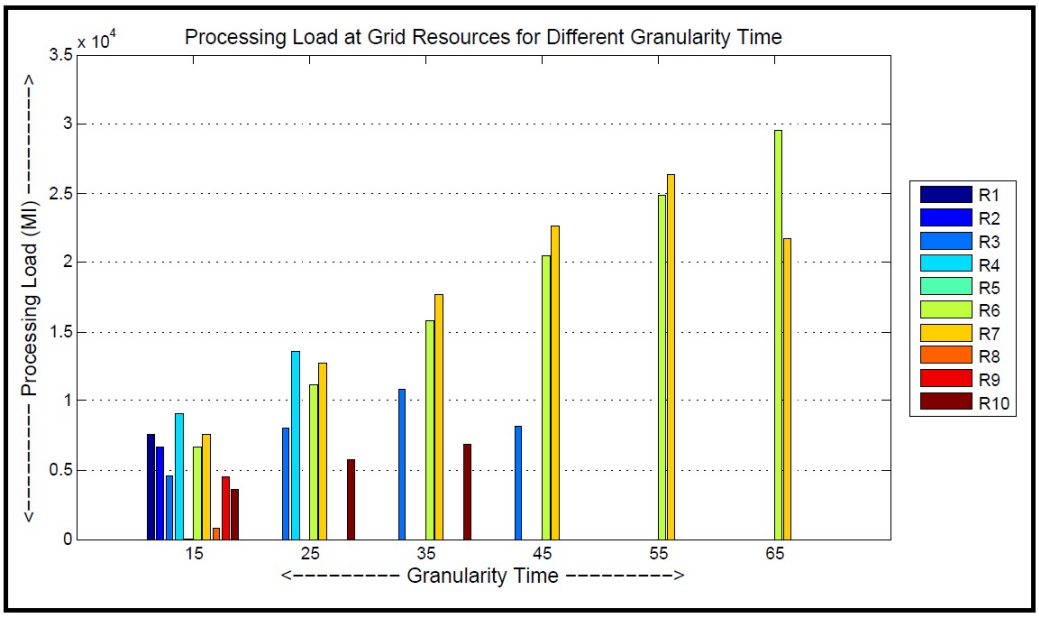


Figure 4.6: Processing load at the grid resources for different granularity time.

From figure 4.6, it is clear that increasing granularity size results poor resource utilization. Therefore, during the job grouping activity granularity time should be determined based on number of jobs, jobs computational requirements and resources computational capabilities to balance the processing load at each resources.

4.3 Summary

In this chapter, we described the implementation details of our simulation and analyzed the simulation results. Simulations are conducted to compare the makespan, computation time, processing cost and resource utilization of proposed DBAJGBS algorithm with DJGBS [10], BAJGBS [19] and GFJS [12] algorithms. From the experiments, it is clear that our proposed Dynamic Bandwidth-Aware Job-Grouping Based Scheduling algorithm reduces the makespan, total computation time of the fine-grained jobs and utilize the resources effectively.

Chapter 5

Conclusions and Future Work

Chapter 5

Conclusions and Future Work

In this thesis we have discussed about the problem of job scheduling in heterecious computational grid, where user submits jobs with a large number of lightweight jobs (requires small processing requirement) and we have tried to find a solution for that problem. We have proposed an efficient Dynamic Bandwidth-Aware Job-Grouping Based Scheduling algorithm. We have got some better performance in terms of processing time and processing cost than some of the job existing grouping based schemes. In the simulation results, it is clear that proposed approach reduces the total processing time and processing cost. However, allocating a large number of Gridlets to one resource will increase the total processing time and processing cost.Therefore, to avoid this situation during job grouping activity, the total number of Gridlet group should be created such that the processing load among the selected resources are balanced.

The proposed approaches have been critically analyzed and few limitations have been observed. These limitations can be studied and refined. Additionally, the simulation environment is semi-dynamic and it can't reflects in the real computational grid environment sufficiently that promotes further research in the proposed area. In future research some more factors like current load of the resource, jobs with a deadline, network delay, QoS (Quality of Service) requirements will be taken into account.

Bibliography

[1] LHC Computing Grid (LCG). http://lcg.web.cern.ch/LCG/.

[2] Ian Foster and Carl Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[3] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35:37–46, June 2002.

[4] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32:135–164, February 2002.

[5] Ian Foster and Carl Kesselman. Grid resource management. chapter The Grid in a nutshell, pages 3–13. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[6] Rajkumar Buyya and Srikumar Venugopal. A gentle introduction to grid computing and technologies. *CSI Communications*, 29(1):9–19, July 2005. Computer Society of India (CSI).

[7] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *JOURNAL OF NETWORK AND COMPUTER APPLICATIONS*, 23:187–200, 1999.

[8] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, 26(4):608 – 621, 2010.

[9] Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, editors. *Grid resource management: state of the art and future trends*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[10] Nithiapidary Muthuvelu, Junyang Liu, Nay Lin Soe, Srikumar Venugopal, Anthony Sulistio, and Rajkumar Buyya. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In *Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44*, ACSW Frontiers '05, pages 41–48, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[11] Ng Wai Keat, Ang Tan Fong, Ling Teck Chaw, and Liew Chee Sun. Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. *Malaysian Journal of Computer Science*, 19(2):117–126, 2006.

[12] Quan Liu and Yeqing Liao. Grouping-based fine-grained job scheduling in grid computing. In *Proceedings of the 2009 First International Workshop on Education Technology and Computer Science - Volume 01*, pages 556–559, Washington, DC, USA, 2009. IEEE Computer Society.

[13] Rajendra Sahu and Anand K Chaturvedi. Article: Many-objective comparison of twelve grid scheduling heuristics. *International Journal of Computer Applications*, 13(6):9–17, January 2011. Published by Foundation of Computer Science.

[14] XiaoShan He, XianHe Sun, and Gregor von Laszewski. Qos guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Technol.*, 18:442–451, July 2003.

[15] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 59:107–131, November 1999.

[16] Yang Gao, Hongqiang Rong, and Joshua Zhexue Huang. Adaptive grid job scheduling with genetic algorithms. *Future Gener. Comput. Syst.*, 21:151–161, January 2005.

[17] Ruay-Shiung Chang, Jih-Sheng Chang, and Po-Sheng Lin. An ant algorithm for balanced job scheduling in grids. *Future Gener. Comput. Syst.*, 25:20–27, January 2009.

[18] Lei Zhang, Yuehui Chen, and Bo Yang. Task scheduling based on pso algorithm in computational grid. *Intelligent Systems Design and Applications, International Conference on*, 2:696–704, 2006.

[19] T.F. Ang and W.K. Ng. A bandwidth-aware job scheduling-based scheduling on grid computing. *Asian Network for Scientific Information*, 8(3):372–277, 2009.

[20] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE)*, 14(13):1175–1220, 2002.

[21] GridSim 5.0 Beta Application Programming Interface(API). http://www.buyya.com/gridsim/doc/api/.